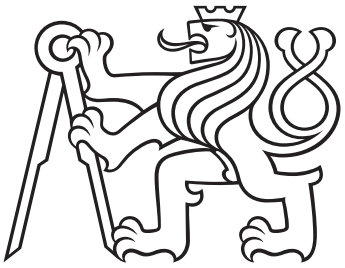
České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačové grafiky a interakce
Obor: Počítačové hry a grafika



Editor virtuálních světů

Virtual world editor

BAKALÁŘSKÁ PRÁCE

Vypracoval: Petr Lhota
Vedoucí práce: Ing. Jaroslav Sloup
Rok: 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Lhota** Jméno: **Petr** Osobní číslo: **483655**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Editor virtuálních světů

Název bakalářské práce anglicky:

Virtual world editor

Pokyny pro vypracování:

Proveďte rešerši metod procedurálního generování terénu (geometrie, materiály) založených na šumových funkcích a fraktálech [1-6].
Na základě nastudované literatury vytvořte editor pro tvorbu map/světů, který umožní procedurálně vygenerovat terén a interaktivně měnit parametry ovlivňující jeho tvar. Implementujte alespoň tři různé metody generování a porovnejte je (např. schopnost generovat kopce, údolí).
Dále navrhněte a implementujte tyto komponenty editoru:
- systém pro generování a mapování materiálů/textur
- generování map (min. zrcadlová, difúzní a normálová mapa)
- sadu nástrojů pro interaktivní editaci terénu
- generování vodní hladiny a jezer
- undo/redo operace
Navrhněte vhodný způsob uložení vygenerovaných světů (formát souborů) a naimplementujte ukládání a načítání tohoto formátu. Formát by měl umožnit oddělené ukládání jednotlivých částí scény (terén, vodní hladina, jezera, atd.).
Funkčnost editoru a jeho komponent demonstrujte alespoň na třech různých scénách. Vygenerované scény porovnejte se skutečnými fotografiemi podobně vypadajících krajin.
Implementaci proveďte v C/C++ s využitím OpenGL.

Seznam doporučené literatury:

[1] David S. Ebert et al.: Texturing & Modeling: A Procedural Approach. Morgan Kaufmann, 2003.
[2] E.Galin, E.Guérin, A.Peytavie, G.Cordonnier, M.-P. Cani, B.Benes, J.Gain: A Review of Digital Terrain Modeling. Computer Graphics Forum, 38(2), 553–577, 2019.
[3] Ryan Vitacion: Procedural Generation of Planetary-Scale Terrains in Virtual Reality. 2019. PhD Thesis. California State University, Northridge.
[4] Tuomo Hyttinen: Terrain Synthesis using Noise. M.Sc. thesis, University of Tampere, 2017.
[5] Jacob Olsen: Realtime Procedural Terrain Generation. 2004.
[6] Jakub Kříž: Multi-Fractal Terrain Generation. Master's thesis, Masaryk University, 2019.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jaroslav Sloup, Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **17.09.2021** Termín odevzdání bakalářské práce: **04.01.2022**

Platnost zadání bakalářské práce: **19.02.2023**

Ing. Jaroslav Sloup podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne .....

.....
Petr Lhota

Poděkování

Děkuji Ing. Jaroslavu Sloupovi za vedení mé bakalářské práce a za podnětné návrhy, které ji obohatily.

Petr Lhota

Název práce:
Editor virtuálních světů

Autor: Petr Lhota

Studijní program: Otevřená informatika
Obor: Počítačové hry a grafika
Druh práce: Bakalářská práce

Vedoucí práce: Ing. Jaroslav Sloup
České vysoké učení technické v Praze

Abstrakt: Tato práce se zabývá procedurálním generováním terénů. Cílem práce je prozkoumat funkčnost a využití fraktálů, šumů a jim podobných algoritmů v počítačové grafice. Je doprovázená editorem s uživatelským rozhraním, který je schopen vytvářet terény, jejich textury, upravovat je a ukládat.
Klíčová slova: grafika, terén, procedurální, generace, šum

Title:
Virtual world editor

Author: Petr Lhota

Abstract: The thesis deals with the issue of procedural generation of terrain meshes. The goal of the thesis is to explore the functionality and usage of fractals and noise functions and other similar algorithms in computer graphics. It is accompanied with an editor for creating, texturing, editing and saving terrain geometries.
Key words: graphics, terrain, procedural, generation, noise

Obsah

Seznam použitých zkratek

fBm Frakční Brownovský pohyb (*Fractional Brownian motion*)

DS Algoritmus diamant čtverec (*Diamond-Square Algorithm*)

RF Algoritmus náhodných poruch (*Random fault algorithm*)

ROAM Optimálně adaptující se mesh v reálném čase (*Real-time optimally adapting mesh*)

LOD Úroveň detailu (*Level of detail*)

Seznam obrázků

[1] Algoritmus náhodných poruch (*Random fault algorithm*)
[2] Frakční Brownovský pohyb (*Fractional Brownian motion*)

Úvod

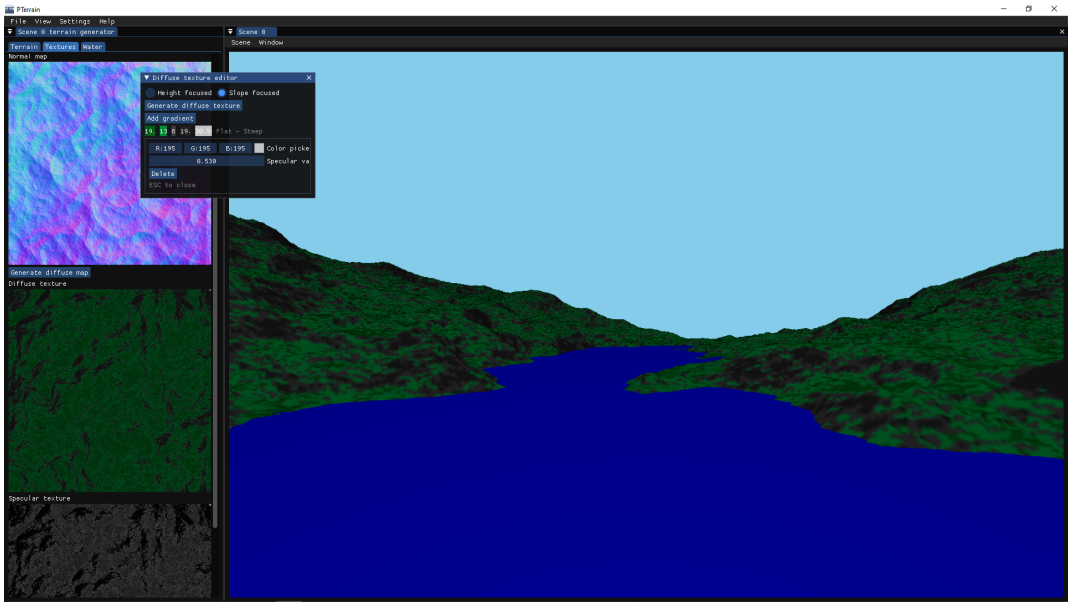
Tato práce se zabývá procedurálním generováním, což je disciplína ve výpočetních technikách, která využívá algoritmického vytváření dat. Častokrát se využívá assetů vytvořených člověkem spojených s algoritmicky vytvořenými objekty a případnou náhodností k dosažení realismu či funkčnosti v daném oboru. V reálném světě se s ním můžeme setkat např. v kombinaci s L-systémy u procedurální generace rostlin, stromů, nebo právě u generování terénu pomocí šumových funkcí či jinými procedurálními algoritmy. Práce je doprovázena prototypem editoru pro generaci terénu napsaného v C++ pomocí vizualizace v OpenGL.

V minulosti se procedurální techniky využívaly pro definování jednoduchých geometrií a barev. Od těchto základů se časem techniky posunuly k pokročilému modelování a hlubšímu texturování objektů. Nejedná se ale jenom o statické objekty. Mezi procedurálně ovládanou geometrii patří například voda, oheň nebo pára. To s sebou nese potřebu matematické zdatnosti programátorů.

S momentálním rapidním vývojem počítačové grafiky se procedurální techniky staly velmi zajímavým a živým nástrojem pro tvorbu těch nejrealističtějších výsledků. Tomu velice dopomáhá schopnost vykreslovat v reálném čase. Nejenže existují velice efektivní nástroje, ale jsou také relativně dostupné například na komerčních počítačích nebo konzolích [1]. Kromě počítačových her se s grafikou tohoto typu můžeme setkat např. ve filmu, reklamě či dokonce při fotografování mobilními telefony s nejnovější technologií.

Kromě vytváření dat se tato disciplína zabývá i optimalizací těchto geometrií. K dnešnímu dni existuje spousta metod zabývající se teselací geometrie či její vizuální reprezentací. Optimalizace se často řeší např. pomocí decimace samotné geometrie nebo vykreslováním pouze jejich zjednodušených částí v momentech, kdy to není poznat.

V následujících kapitolách se práce bude zabývat principy procedurálního generování a existujícími metodami jeho použití v praxi, konkrétně pak v tvorbě šumových funkcí a s nimi spojenými algoritmy. Na základě těchto poznatků byla vyvinuta aplikace PTerrain ve formě editoru pro tvorbu virtuálních světů (obr. 1), kterým se práce zabývá ve své praktické části.



Obrázek 1: Náhled aplikace PTerrain.

Kapitola 1

Procedurální generování

Kapitola pojednává o historii a hlavních charakteristikách procedurálního generování. Je zde popsáno, v jakých případech je vhodné procedurální generování používat a jak maximalizovat jeho účinnost. Dále představuje již vyvinuté aplikace moderní doby.

1.1 Historie

Procedurální generování dat se poprvé objevilo u rogue-like her (podle počítačové hry *Rogue (1980)*). Motivací vývojářů bylo vytvořit hru, která bude s každým startem trošku jiná i přestože bude mít stejné mechaniky. Další motivací bylo i to, že přenašeče dat v té době byly velice kapacitně omezené, a tak by takové množství úrovní nebylo možné uchovat na jakémkoliv médiu. Už v letech 1985 se začaly objevovat články o procedurálním generování a jeho potenciálním využití. Darwin R. Peachey ve svém článku *Solid Texturing* odkazuje na možnost vytvářet 3D textury pomocí procedurálních technik [2]. Později se procedurální přístupy vyskytovaly mnohem častěji - například herní tituly *Diablo* (1996), *The Elder Scrolls II: Daggerfall* (1996).

Procedurální modelování však historicky nebylo použito jen ve videohrách, ale i v animovaných filmech. Studio Pixar vyvinulo svůj vlastní animační engine *RenderMan®*, který nabízel procedurální techniky pro tvorbu textur a materiálů [2].

1.2 Procedurální přístup k datům

Podle profesora Iana Parberryho musí procedurální generování mít tři důležité vlastnosti. První z nich je nenáročnost provozu, neboli nezávislost na nejnovějších technologiích. Poté je nutné, aby generování bylo náhodné a strukturované tak, aby vytvářelo obsah, který je různorodý a zajímavý. Nakonec by se s vygenerovanými daty mělo zacházet jednoduše a intuitivně [3].

Nicméně tento přístup generování a modelování se používá při tvorbě velice komplexních modelů. Těmito modely nejčastěji bývají terény, stromy, mraky, ale také třeba člověkem vytvořené objekty, jako např. stavby.

1.2.1 Výhody procedurálního přístupu a kdy ho používat

Pro tvorbu těchto objektů potřebujeme poměrně malý objem dat, který popisuje charakteristiky daného objektu. Model je následně vytvořen procedurou, která používá pseudonáhodná čísla. Tímto způsobem můžeme vytvářet struktury, ve kterých se vyskytují podobné modely, ale žádný z nich není stejný (např. lesy, vlasy, vegetace). Procedurální metody tedy jistým způsobem amplifikují původní data [4].

Hlavním důvodem je unikátnost založená na individuálních zkušenostech uživatelů. Díky procedurálnímu generování vznikají nové interakce, nepředvídatelnost a živoucí systémy. To u některých videoher prohlubuje faktor zábavy.

Další výhodou procedurálního generování je produkce velkého množství různorodých dat, tedy znovupoužitelnost. Není to ale pouze recyklovatelnost v rámci jednoho programu, ale jedná se i o reprodukci kódu. Simulátor ohně může v jiném programu simulovat roznášení nákazy.

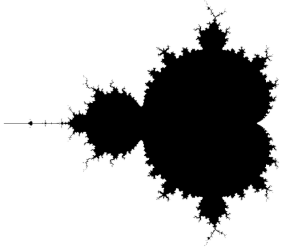
Na druhou stranu, programy (resp. videohry) založené na procedurálním generovaní nutně nemusí vždy pracovat tak, jak bylo původně zamýšleno. Je prakticky nemožné pro testery procházet každou iteraci procedurálního obsahu. Příklad: chyba se může vyskytnout v 0,1% případů, se kterými se setká jenom hrstka uživatelů, ale chybovost může být i vyšší [5].

1.3 Fraktály

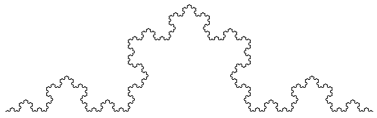
Fraktály jsou objekty tvořeny rekurzivní transformací prvků. Teoreticky to jsou nekonečně detailní objekty. Podle B. B. Mandelbrota, který tyto objekty pojmenoval (1982), je fraktál "hrubý nebo rozkouskovaný tvar, který může být rozdělen na části, které jsou zmenšenými kopiemi jeho celku"[6]. Tato definice naznačuje důležitý faktor soběpodobnosti, který je jednou z charakteristik fraktálů. Na této definici se však většina autorů rozchází. Shodují se ale na jedné věci, a to, že fraktální vzory jsou charakterizovány fraktálními dimenzemi, které sice určují komplexitu fraktálu, ale nedefinují, jak daný fraktál vypadá [7].

Jednoduchá heuristická definice fraktálu jako geometricky komplexního objektu, jehož komplexita se zvětšuje přes repetice dané formy [1], by pro tuto práci měla být postačující.

Fraktály jsou v matematice a počítačových vědách prakticky velice mladou disciplínou. Vyvíjely se společně s počítači už jen díky tomu, že počítače velice usnadnily studium fraktálů pomocí grafického zobrazení. Bez pomoci počítačů jsou fraktály moc složité na vykreslování a prostudování. Mezi historicky nejslavnější fraktály patří Mandelbrotova množina (obr. 1.1) a Kochova křivka (obr. 1.2).



Obrázek 1.1: Mandelbrotova množina [8].



Obrázek 1.2: Kochova křivka [9].

1.3.1 Použití fraktálů v praxi

Mějme geometrii G_0 a transformaci T, pak každá iterace rekurzivní transformace vygeneruje detailnější geometrii. Tuto skutečnost označíme vztahem

$$G_i = T(G_{i-1}).$$

V počítačové grafice ovšem chceme, aby tento průběh byl nějakým způsobem terminován. Rekurzi proto spouštíme do určité hloubky a dostáváme tak konečný detail geometrie. Ovšem, všechny fraktály nejsou ideální. Například právě Mandelbrotova množina (obr. 1.1) je sice velice krásná, nicméně pro počítačovou grafiku je víceméně nepoužitelná - na rozdíl pro jiné odvětví, např. opodstatnění teorií chaosu díky bifurkačnímu diagramu, který je součástí této množiny, pokud se na ni podíváme ve třetí dimenzi z boku.

1.3.2 Vytváření terénů

Pokud se podíváme na jakýkoliv zemský terén, povšimneme si, že je nekonečně detailní a téměř náhodný. Proto vyžadujeme, aby metody jejich vytváření splňovaly tyto požadavky. Právě proto jsou pro jejich tvorbu zpravidla fraktály a šumové funkce vhodné.

1.4 Již existující nástroje pro vytváření terénů

K dnešnímu dni existuje poměrně široká škála programů zabývající se touto tématikou. Mezi ty nejznámější patří Terragen (obr. 1.3), Bryce, World Machine (obr. 1.4) a Vue. Oproti známějším programům pro manipulaci s geometrií a materiály, jako je např. Blender nebo jeho alternativy (Maya, Cinema4D apod.), jsou tyto programy postaveny tak, aby práce se složitou geometrií byla co nejjednodušší.

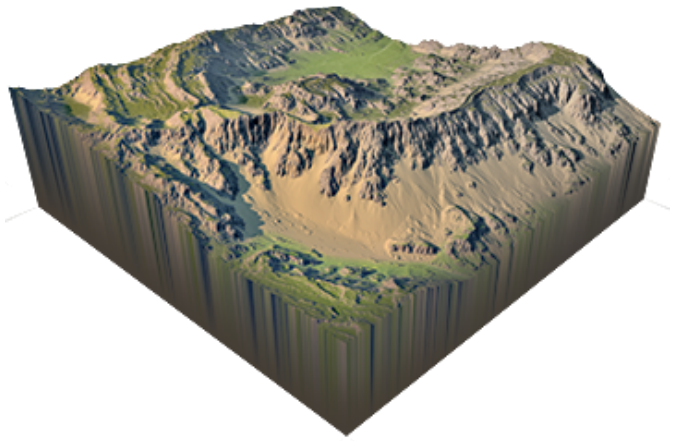
Vue, Bryce a Terragen oproti Blenderu využívají pouze data výšek jednotlivých souřadnic, zatímco Blender si uchovává 3 floaty určující jejich souřadnici (mimo všechny ostatní informace týkající se sousednosti). To znamená, že terény s rozlišením 1024x1024 jsou v těchto programech triviální, zatímco v Blenderu jsou to velice komplexní objekty. S tím souvisí i optimálnější algoritmy týkající se trasování v rámci této geometrie [10].

Kromě toho jsou tyto typy programů vytvořeny právě pro dané operace, tudíž mají vlastní metody, jak docílit realismu. Vue i Terragen používají velice mocné metody k vytváření atmosféry, ať už se jedná o mlhy, mraky a jiné. Právě z tohoto důvodu se tyto programy používají i ve filmovém průmyslu. Terragen byl například použit ve známém úvodu do filmů Paramount Pictures, nebo např. ve filmech Star Wars: The Last Jedi (2017) a Thor: Ragnarok (2017).

World Machine je oproti tomu určený specificky ke generování vysoce detailních terénů. Tento editor prezentuje pro referenci pouze preview reálného výsledku v nižším rozlišení. Pomocí tohoto programu je možné také velice reálně simulovat erozi na terénech z elevačních dat dostupných např. na Google Maps.

Terragen rozlišuje procedurálně generované terény a *heightfields*, tedy výšková pole. Zatímco procedurálně generovaný terén je nekonečný a nekonečně detailní, výšková pole jsou konečná a definují vzhled celé planety, resp. jejího úseku. Nabídka algoritmů pro generaci terénů Terragen v0.9 z roku 2005 obsahuje *Subdivide and Displace (Midpoint Displacement), Perlin Noise* a jeho multifraktální verze.

Obrázek 1.3: Terén vytvořený v Terragenu [11].



Obrázek 1.4: Terén vytvořený ve World Machine [12].

Kapitola 2

Šumové funkce

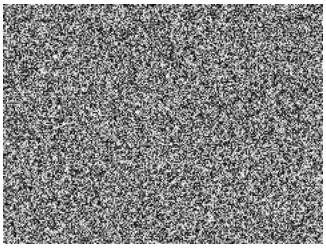
Jedna z největších výzev vědy počítačové grafiky je dosažení co možná nejvyšších detailů. Je totiž důležité, aby taková data využívala co nejmenší množství paměti a bylo jednoduché je vytvořit. Právě proto jsou upřednostňovány šumové funkce, které tyto vlastnosti splňují. Konkrétně ke generování různorodých procedurálních výškových map a textur používáme nejčastěji právě heterogenní a primitivní funkce nazývané šumy. Tato kapitola definuje, co šumové funkce jsou a jak se používají. Také demonstruje typy nejčastějších a nejznámějších šumových funkcí využívaných v počítačové grafice [13].

2.1 Definice

Jsou to stochaistické[1], většinou spojité funkce, které slouží k tomu, abychom přerušili monotónnost vzorů, které by jinak byly pravidelné. Při použití slova náhodný a stochaistický je myšleno pseudonáhodný (téměř náhodný). Při tvorbě takovýchto textur není žádoucí, aby vytvořená data byla opravdu náhodná [1].

2.2 Bílý šum

Bílý šum (*white noise*, obr. 2.1) je názorným úkazem stochaistické textury. Je to zdroj náhodných čísel, která jsou rovnoměrně rozprostřena, tedy mají stejnou spektrální hustotu, aniž by mezi nimi existovala jakákoliv korelace.



Obrázek 2.1: Bílý šum [14].

[1]Stochastický znamená náhodný, nahodilý. Protikladem je deterministický.

2.3 Šumové funkce založené na mřížkách

Mřížové šumy (*lattice based noises*) jsou nejčastější implementací šumů pro texturální aplikace. Generování začíná jedním nebo více rovnoměrně distribuovanými náhodnými čísly na mřížce, která je definována pozicemi jednotlivých rovnoměrně rozestavěných vrcholů. Tyto hodnoty jsou koherentní. Mezi těmito vrcholy se po vygenerování náhodných čísel provede interpolace[2]. Abychom docílili korektního generování, musíme pro každý bod vytvořit hodnotu přes hashovací funkci, která bude obsahovat náhodné permutace čísel, tedy nějakou pseudonáhodnou sekvenci [1].

2.4 Hodnotové šumy

Hodnotové šumy (*value noise*, obr. 2.2) jsou mřížové šumy. Fungují tak, že do vrcholů jsou přidány pseudonáhodné hodnoty (stejně jako u bílého šumu), které jsou následně interpolovány se svými sousedy. Může se používat lineární interpolace, nicméně vytvořené šumy pak vypadají víceméně jako čtverce (resp. krychle) se znatelnými artefakty. Jejich derivace není spojitá, čímž je způsobena tvorba hran mezi jednotlivými interpolovanými hodnotami.

2.4.1 Catmull-Rom interpolace

Proto je lepší použít kubickou Catmull-Rom spline interpolaci. V jedné dimenzi vypadá jako:

$$cr_{cub}(x) = f_{i-1}w_0(\alpha) + f_i w_1(\alpha) + f_{i+1}w_2(\alpha) + f_{i+2}w_3(\alpha), \tag{2.1}$$

kde $x = i + \alpha$ pro $i \in Z$ a $\alpha \in [0, 1)$, kde α je zlomkový zbytek. Váhy korespondující s Catmull-Rom spline jsou následující [15]:

$$\begin{aligned} w_0(\tau) &= -\frac{1}{2}\tau^3 + \tau^2 - \frac{1}{2}\tau \\ w_1(\tau) &= \frac{3}{2}\tau^3 - \frac{5}{2}\tau^2 + 1 \\ w_2(\tau) &= -\frac{3}{2}\tau^3 + 2\tau^2 + \frac{1}{2}\tau \\ w_3(\tau) &= \frac{1}{2}\tau^3 - \frac{1}{2}\tau^2 \\ \tau &\in [0, 1) \end{aligned}$$

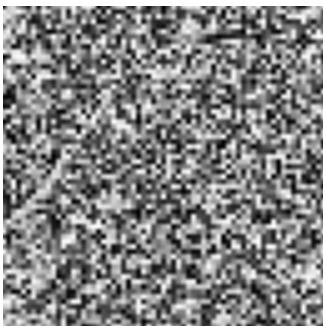
Catmull-Rom spline interpolace je založena na všech sousedících bodech ve všech dimenzích těchto šumů. V dimenzi n je 4^n kontrolních bodů, tedy ve 2 dimenzích má kontrolních bodů 16, ve 3 dimenzích má kontrolních bodů 64. Jedná se proto o velmi náročnou operaci [1]. Mezi nejpoužívanější alternativy pro výpočet hodnotových šumů patří kvadratická a kubická B-spline interpolace či Wienerova interpolace.

[2] Jen a proto, že množina hodnot nemůže nikdy obsahovat frekvence vyšší než Nyquistova frekvence tohoto intervalu, tedy by nám vznikal alias.

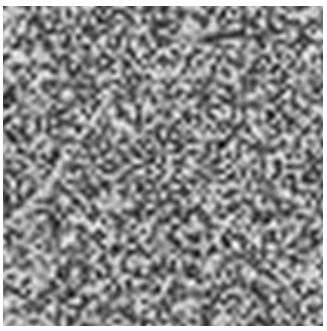
2.4.2 Kubický hodnotový šum

Pro hodnotový šum (obr. 2.3) je vhodných několik typů interpolací. Záleží však také, kolik času pro jeho tvorbu je k dispozici. Mimo lineární se používá kosinová nebo právě kubická interpolace, jejíž výpočet trvá velmi dlouho, ale přináší perfektní výsledky.

Pokud však chceme vytvářet výškové mapy v reálném čase, musíme používat lineární interpolace [16]. Na následujících příkladech lze zpozorovat, že kubický hodnotový šum má mnohem plynulejší přechody mezi jednotlivými hodnotami, nicméně stále zachovává vysokou hladinu detailu.



Obrázek 2.2: Hodnotový šum.



Obrázek 2.3: Kubický hodnotový šum.

2.5 Gradientní šumy

Oproti hodnotovým šumům se gradientní šumy vytváří metodou pseudonáhodných gradientů (tedy vektorů) v mřížce, které jsou následně interpolovány pro získání hodnot mezi buňkami [17]. Pro interpolaci se využívají gradienty všech vrcholů jedné buňky spíše než kubická interpolace přes sousedních vrcholy [1].

2.5.1 Perlinův šum

Perlinův šum (*Perlin noise*, obr. 2.4), metoda Kena Perlina z roku 1985, je mřížový šum gradientního typu. Je to pravděpodobně nejpoužívanější typ šumu - je rychlý, různorodý a používá velice málo paměti. Je velice podobný hodnotovému šumu, nicméně Perlinův šum místo výškové hodnoty využívá gradient pro každý bod.

Obrázek 2.4: Perlinův šum.

Pro jeho tvorbu použijeme dvě mřížky: tabulku gradientů a tabulku permutací. Tabulka gradientů je většinou několikanásobně menší než permutační tabulka, která obsahuje hodnoty od 0 do $n - 1$. Ta je pak následně náhodně proházena.

Pro každý pixel poté aplikujeme tento algoritmus [16]:

1. Najdeme sousedící body a jejich gradienty.
2. Vypočítáme skalární součin gradientu a vzdálenosti vektoru od daného bodu.
3. Provedeme lineární interpolaci mezi těmito skalárními součiny.
4. Aplikujeme na výsledné hodnoty útlumovou funkci

$$t = 3t^2 + 2t^3.$$

Složitost tohoto algoritmu je $O(n \cdot 2^n)$, kde n je dimenze.

2.5.2 Simplexový šum

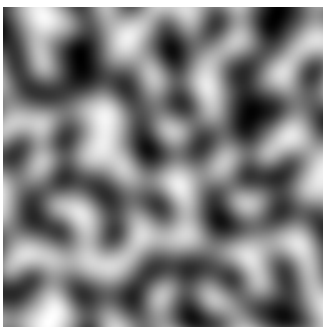
Simplexový šum (*simplex noise*, obr. 2.5) vynalezl Ken Perlin v roce 2001. Důvodem byla nedostatečná efektivita Perlinova šumu v dimenzích $n > 3$ [16] [18]. Nevýhodou Perlinova šumu je používání interpolací. Na rozdíl od Perlinova šumu je simplexový šum založen na sčítání a následné aplikaci útlumové funkce, díky čemuž má polynomiální složitost $O(n^2)$.

Implementace a porovnání

Namísto hyperkrychlí simplexový šum využívá simplexy. *Simplex* je útvar s nejmenším možným počtem vrcholů v této dimenzi, tedy tvar s $n + 1$ vrcholy, kde n je dimenze (v 2D trojúhelník, v 3D čtyřstěn). V dimenzi $n = 15$ by Perlin noise vygeneroval tabulku o 32768 bodech, zatímco simplex noise vygeneruje pouze 16 bodů. Pro tyto body je potom nutné spočítat opět skalární součiny a aplikovat novou, upravenou poloměrovou útlumovou funkci ve tvaru

$$t = 6t^5 - 15t^4 + 10t^3$$

na každý z těchto vrcholů [16] [19].



Obrázek 2.5: OpenSimplex2S z knihovny FastNoise.

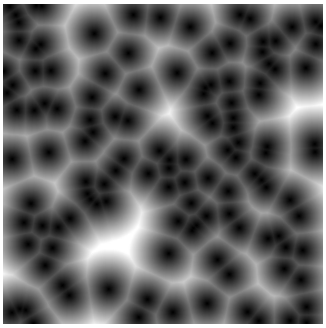
Tento šum je oproti Perlinově šumu velice kvalitní a rychlý, tudíž se jeví jako *nejlepší* z dosud použitelných šumů. Na druhou stranu je ale jeho implementace velice náročná. Velikou překážkou pro vývojáře je také patent, který zamezuje jeho běžnému použití.

2.6 Šumové funkce založené na bodech

Kromě šumů založených na mřížkách můžeme definovat šumy založené na klíčových bodech volně rozložených v prostoru. Takové funkce nejsou zobrazovány pravidelnými, nicméně mnohdy velmi podobnými tvary. Jsou taktéž koherentní. Mezi nejznámější patří Worleyho šum.

2.6.1 Worleyho šum

Worleyho šum (*Worley noise*, obr. 2.6) vytvořený Stevem Worleym v roce 1996 je šum určený hlavně pro tvorbu procedurálních textur. Poměrně slibně simuluje textury kamene, vody nebo biologických buněk. Z tohoto důvodu se nazývá *buňkový šum* (*cellular noise*). Navíc buňková je i texturová báze, na které je založen. Jeho implementace závisí na vzájemné vzdálenosti kontrolních bodů jednotlivých buněk, k čemuž se využívá pole vzdáleností.



Obrázek 2.6: Worley noise z knihovny FastNoise.

K jeho vytvoření je definovaná bázová funkce s náhodnými klíčovými body. Pro každé umístění bodu x existuje klíčový bod, který leží blíže x než kterýkoliv jiný. Tuto nejmenší vzdálenost můžeme označit jako $F_1(x)$. Při změně umístění x se tedy mění i F_1. Nicméně se může stát, že F_1 bude stejná pro více bodů najednou, ale to algoritmu nevadí, protože v tu chvíli nezáleží na který z bodů F_1 ukazuje. Dále jsou definovány funkce F_i, kde F je i-tá nejbližší vzdálenost od jednotlivých bodů. F je také vždy spojitá. Pro texturování stačí na F uvést mapovací metodu. Nejprimitivnější metodou je vybarvit F_1 pomocí spline. Pokud jsou F_1, F_2 a F_3 použity v mapovací funkci, jedním z možných výsledků jsou šedé gradienty, např. v kombinaci $F_2 - F_1$. Ve výsledku mohou být zpozorována místa, kde $F_2 - F_1 = 0$. Taková místa naznačují Voronoiovu hranici (obr. 2.6) [20].

Kapitola 3

Procedurální algoritmy pro tvorbu terénů

V této kapitole jsou představeny algoritmy, které vytváří fraktální plochy. Tyto algoritmy patří mezi prakticky nejpoužívanější. Některé z nich jsou použity v implementované aplikaci. Další algoritmy lze nalézt např. v [1].

3.1 Frakční Brownův pohyb

Frakční Brownovský pohyb (*fractional Brownian movement*, dále fBm), alternativně fraktální Brownovský pohyb, je zobecnění původního Brownovského pohybu, tedy pohybu částic na nějakém médiu [21]. Procedurální fBm se v praxi aplikuje pomocí počtu předem definovaných oktáv, lakunarity[1] a inkrementálního parametru fraktálu H.

Algorithm 1: Evaluace fBm v bodě [1]

oktávy = *double*
lakunarita = *double*
hodnota = 0.0
Noise() //funkce tvořící šum
Noise3() //funkce použitá pro prostorový šum
for *(i = 0; i < oktávy; i++)* **do**
  $hodnota = hodnota + Noise(bod) \cdot lakunarita^{-H \cdot i}$
  $bod* = lakunarita$
end
$zbytek = oktávy - (int)oktávy$
if *zbytek* **then**
  $hodnota = hodnota + zbytek \cdot Noise3(bod) \cdot lakunarita^{-H \cdot i}$
end
Result: hodnota v bodě

Tato velice jednoduchá metoda nám zaručí potlačení chaosu v šumu. Pomocí parametru H tento chaos můžeme ovládat. Když $H = 1$, pak je výsledek fBm velice uhlazený, na druhou stranu když $H \to 0$, tak se výsledný šum podobá spíše bílému šumu (obr. 2.1).

[1] Lakunarita, odvozeno z angl. "lake", je mezera mezi po sobě jdoucími frekvencemi

3.2 Multifraktály

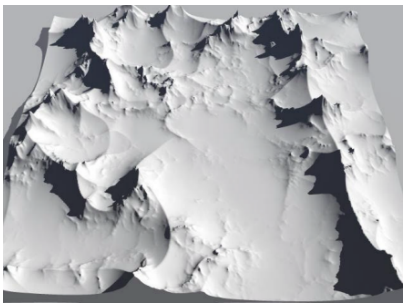
Multifraktály jsou kombinací různých typů fraktálů. To zaručuje jejich heterogennost. Pomocí nich je možné generovat výškové mapy, které obsahují nejen roviny a údolí, ale i hory.

Mezi typy multifraktálů patří například hybridní multifraktály nebo zvrásněné multifraktály.

3.2.1 Zvrásněné multifraktály

Zvrásněný brownův pohyb (ridged fBm, obr. 3.1) zvýrazní hrany daného šumu. Používá se pro tvorbu pohoří v kombinaci s rovinami či vln na moři (resp. oceánu).

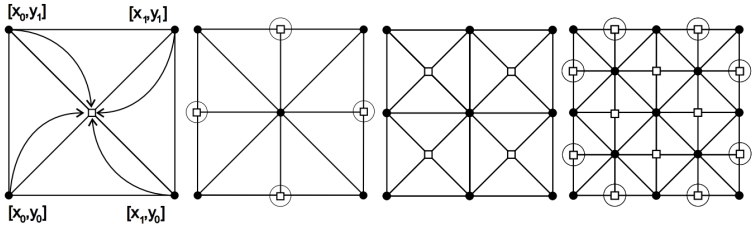
Jeho principem je, že z hodnoty šumové funkce vypočítá její inverzní hodnotu, tedy $1 - abs(noise)$.



Obrázek 3.1: Terén vygenerovaný pomocí zvrásněného multifraktálu [22].

3.3 Metoda náhodného přesouvání středního bodu

Metoda náhodného přesouvání středního bodu (*random midpoint displacement*) je procedurální algoritmus pro generaci fBm. Pomocí rekurzivního dělení úseček (resp. čtverců, obr. 3.2) na polovinu a posouvání středních bodů o náhodné číslo δ vytvoříme geometrii.



Obrázek 3.2: Rekurzivní dělení čtverců [23].

Nicméně, abychom dodrželi vzor fBm, musí být δ_1 náhodné číslo s normalizovaným Gaussovým rozložením $N(0,1)$ a v $i-$té iteraci musí být δ_i náhodné číslo též s Gaussovým rozložením, které má oproti σ^2 modifikovaný rozptyl v závislosti na H a i [23].

$$\sigma_i^2 = \frac{1}{2^{2 \cdot H(i+1)}} \cdot \sigma^2 \tag{3.1}$$

V praxi se pomocí této skutečnosti využívá pro výpočet posunutí bodu

$$\delta = H \cdot Gauss(0,1) \cdot |A,B|,$$

kde H je Hurstův exponent, Gauss(0,1) je výše zmiňované náhodné číslo s normalizovaným Gaussovým rozložením a |A,B| je vzdálenost bodů A a B [23].

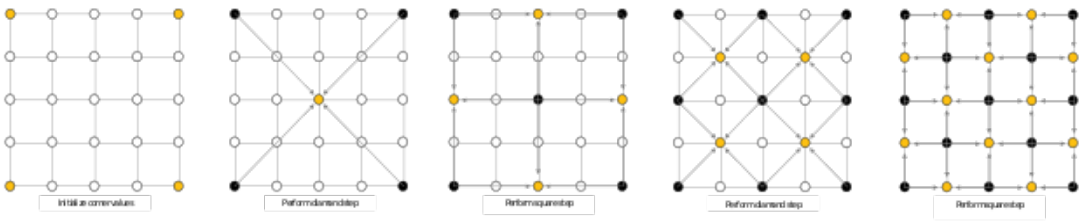
Hodnoty první iterace (inicializační vrcholy čtverce) mají největší vliv na tvar výsledné geometrie.

3.4 Algoritmus diamant-čtverec

Algoritmus diamant-čtverec (*diamond-square algorithm*, dále DS[2]) je vylepšením metody přesouvání středního bodu (3.3). Taktéž generuje plochu fBm. DS funguje pouze pro objekty se stejnou výškou a šířkou, které se musí rovnat $2^n - 1$. Rohové body se nastaví na inicializační hodnoty, které ovlivňují výsledný tvar výškové mapy, a následně se provede algoritmus, který alternuje mezi:

1. Diamantovým krokem: provedení součtu hodnot v rozích čtverce, následně vypočítání průměru a přičtení náhodné hodnoty (obr. 3.3-2,4).
2. Čtvercovým krokem: provedení součtu hodnot v rozích otočeného čtverce o 45°, následně vypočítání průměru a přičtení náhodné hodnoty (obr. 3.3-3,5).

Přičítaná náhodná hodnota je vybrána z normálního rozdělení a násobena mitigací 2^{-Hi}, kde $H \in [0,1]$ je koeficient určující hrubost terénu ($H \to 0$ pro nejhrubší terén) a i-té iterace, resp. kroku. To opět zaručí modifikovaný rozptyl rovný 3.1.



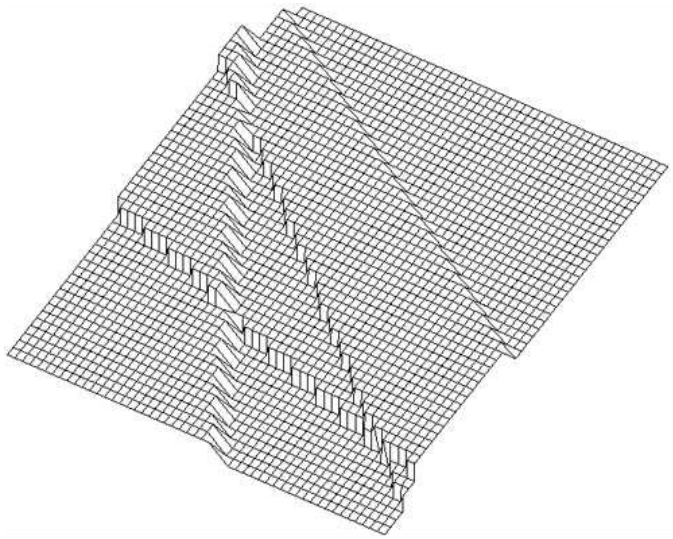
Obrázek 3.3: Diamantový a čtvercový krok [24].

3.5 Algoritmus náhodných poruch

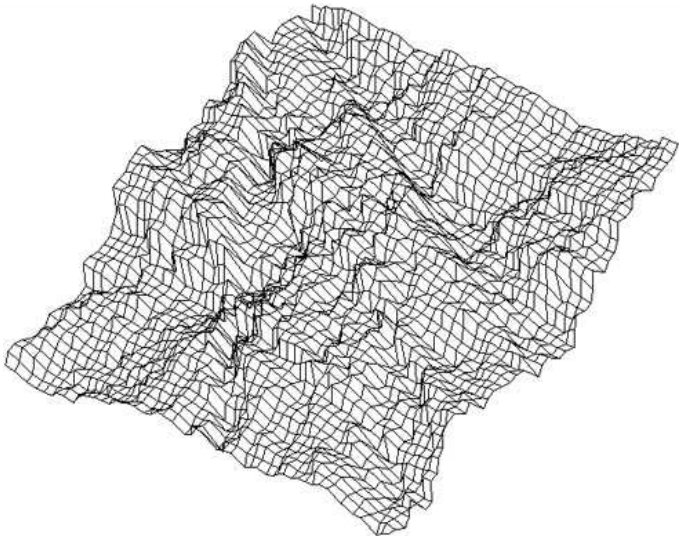
Algoritmus náhodných poruch (*Random fault algorithm*, obr. 3.4 a 3.5) generuje plochu fBm. Je aproximací tvorby reliéfu způsobenou zlomy tektonických desek a následnými zdvihy podél nich [25]. Algoritmus na plochý terén aplikuje v cyklu iterací následující kroky:

1. Vybere dva body z mřížky a proloží jimi přímku.
2. Zvýší (resp. sníží) body na jedné straně přímky o náhodné číslo z Gaussova rozdělení δ_i, na druhé straně přímky všechny body o tuto hodnotu sníží (resp. zvýší).

[2] Algoritmus diamant čtverec (*Diamond-Square Algorithm*)



Obrázek 3.4: Plocha po čtyřech iteracích RF [23].



Obrázek 3.5: Vygenerované fBm pomocí RF [23].

Kapitola 4

Návrh

V této kapitole jsou navrženy metody, kterými se bude výsledná aplikace řídit. Jsou zde popsány způsoby vytváření geometrií a jejich úpravy pomocí výškových map. Jsou zde taktéž definovány formáty, ve kterých je vhodné s objekty pracovat a do jakých je ukládat.

4.1 Terén

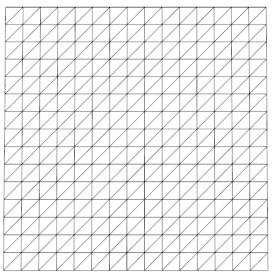
Schopnost generovat terény je hlavním cílem výsledné aplikace. V editoru by uživatel měl mít plnou kontrolu nad tím, jak upravovat dané terény včetně jejich příslušných textur. Přístup obarvování meshe pomocí textur namísto materiálů byl zvolený z důvodu jednodušší upravování a přehlednější práci v prostředí editoru.

4.1.1 Geometrie terénu

Terén je v základu jednoduchá mřížka vrcholů spojená trojúhelníkovými ploškami (obr. 4.1). Každý z vrcholů je unikátní a označuje příslušné texturové souřadnice a normálový vektor příslušící vrcholu[1]. Souřadnice vrcholu jsou definovány jako

$$vrchol = (\mathbf{x}, \mathbf{z}, \mathbf{y}),$$

kde x je šířka, z je hloubka a y je výška. V každém terénu o šířce w a výšce h se pak nalézá $w \cdot h$ vrcholů a $2 \cdot (w-1) \cdot (h-1)$ plošek. U méně detailních terénů je znatelný rozdíl ve tvaru, který závisí na souřadnicové orientaci trojúhelníkových plošek, nicméně u vysoce detailních terénů je tento faktor zanedbatelný.



Obrázek 4.1: Geometrie terénu 16×16 s implementovanou topologií.

[1] z důvodu reprezentace v OpenGL

Uživatel má možnost navolit výšku a šířku terénů a vzdálenost jednotlivých bodů. Je informován o rozlišení a velikosti výsledné geometrie. Změnou hloubky jednotlivých vrcholů se mění tvar terénu. Pro tvorbu terénů byly zvoleny tyto přístupy:

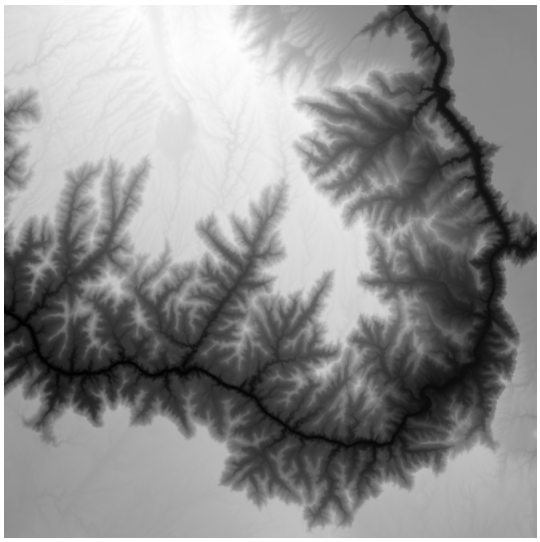
1. Plochý terén.
2. Pomocí algoritmu diamond-square (3.4).
3. Pomocí metody náhodných poruch (3.5).

4.1.2 Textury terénu

Každý objekt terénu je namísto materiálu vlastníkem těchto textur - výškové mapy, difuzní textury, normálové textury a spekulární textury. Ty jsou editorem automaticky generovány podle uživatelem zadaných parametrů, difuzní a spekulární textury jsou uživatelem upravitelné. Pro ukládání a načítání textur jsou povoleny formáty *.jpg*, *.png*, *.tga* a *.bmp*.

Výšková mapa

Výšková informace jednotlivých bodů je reprezentována v textuře výškové mapy. Ta je jednokanálová (ve stupních šedi). Každou výškovou mapu je možné nahrát ze souboru, který musí respektovat rozměry terénu, nebo vytvořit přímo v prostředí editoru. Každá výšková mapa generovaná editorem si také pamatuje předchozí stavy, aby se uživatel mohl vracet pomocí undo (resp. redo) operací.



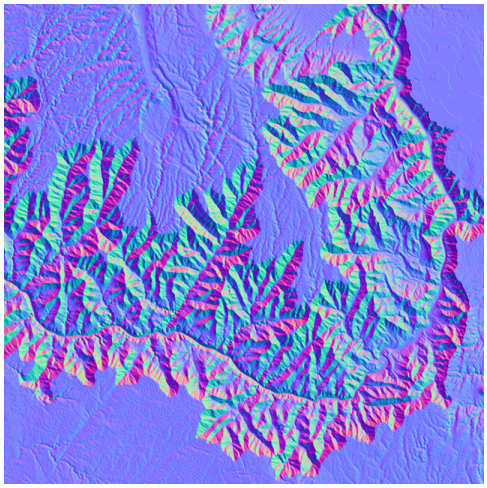
Obrázek 4.2: Výšková mapa východního cípu řeky Colorado v Grand Canyonu [26].

Normálová textura

Je generována aplikací automaticky pomocí informací o normálových vektorech jednotlivých vrcholů. Protože normalizované normálové vektory mají hodnoty v intervalu $[-1, 1]$, barevný posun této textury vypadá následovně (obr. 4.3)

$$(\mathbf{r}, \mathbf{b}, \mathbf{g}) = (\mathbf{v_n} + (\mathbf{1}, \mathbf{1}, \mathbf{1})) * 0.5 * 255. \tag{4.1}$$

To zaručí, že všechny hodnoty se pohybují v intervalu $[0, 255]$. Hodnoty ležící v $[0, 127]$ jsou záporné, kdežto hodnoty ležící v $[128, 255]$ jsou kladné.



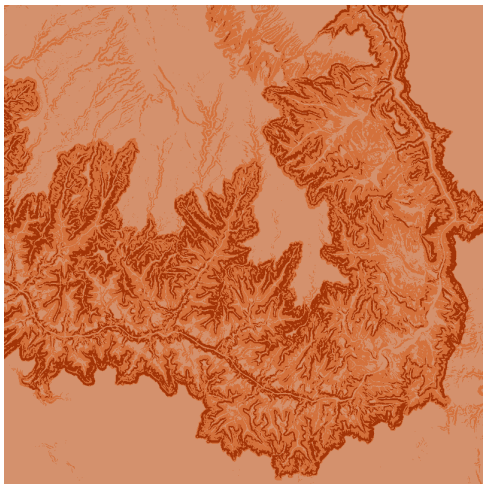
Obrázek 4.3: Normálová mapa z terénu na obr. 4.2. Dodržuje barevné schéma z 4.1.

Difuzní textura

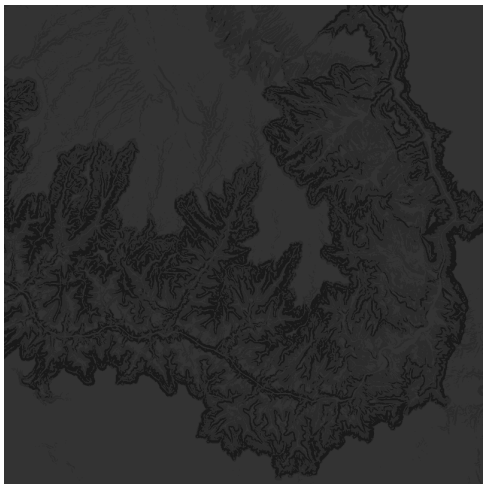
Definuje obarvení terénu v rámci jedné textury (obr. 4.4). Je v kanálech RGB, určená hodnotami od 0 do 255. Uživatel ji může generovat přes dva zvolené způsoby: barvení podle výšky, barvení podle sklonu plochy. Pro oba tyto způsoby jsou navrženy gradienty, které definují interval výšek, resp. úhlu od vektoru směrujícího vzhůru.

Spekulární textura

Spekulární textura určuje odrazivost světla od vrcholu. Má pouze jeden kanál a její hodnoty se pohybují v intervalu $[0, 255]$. Je generována současně s difuzní texturou (obr. 4.5).



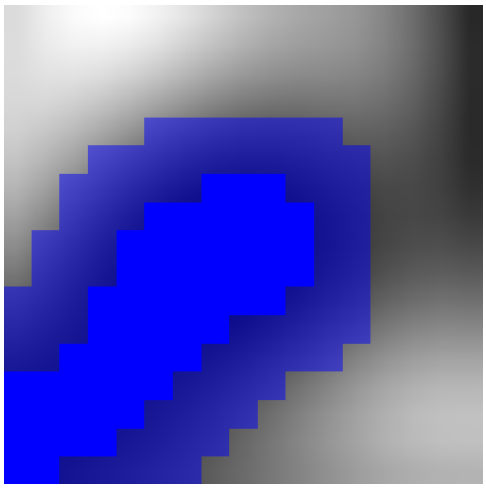
Obrázek 4.4: Obarvení terénu pomocí sklonu.



Obrázek 4.5: Obarvení spekulární textury pomocí gradientů.

4.1.3 Vodní plochy

Vodní plochy jsou podobjekty terénu vytvářené pomocí speciální vodní textury udržované v aplikaci. Kliknutím na výškovou mapu terénu uživatel vytvoří simulaci proudu vody, který steče do lokálního minima po ploše terénu. V momentě, kdy narazí na minimum nebo bod, na kterém se již vodní hladina nachází, obarví tento bod modrou barvou a vytvoří přesahující ohraničení, které je znázorněno průhlednou barvou (obr. 4.6). Modré body jsou ty, které jsou viditelné na povrchu terénu, zatímco poloprůhledné body jsou ty, které jsou skryté pod terénem a protínají jeho plochu. Zaručují ale, aby mezi okrajem vodní plochy a terénem nevznikaly mezery.



Obrázek 4.6: Vyznačná mesh na textuře 17x17.

4.1.4 Ukládání a načítání

Pro ukládání a načítání souborů je zvolen jeden z nejčastějších formátů pro ukládání souborů *wavefront .obj*. Do tohoto formátu je přidán komentář *"OBJ file created by or for PTerrain:"*, za kterým následuje další komentář s názvem terénu, jeho šířkou a výškou.

4.2 Scéna

Scéna je klíčovým držitelem kontextu objektů. Každá scéna je vlastníkem ukazatelů na framebuffer, kameru, shader a objekty, které jsou součástí scény. Framebuffer má každá scéna vlastní. Pro jednoduchost si scéna nedrží žádné informace o pozicích jednotlivých objektů. Není to pro jednoduchost programu nutné, data jsou totiž upravována přímo.

4.3 Engine

Engine je samostatnou a oddělenou částí výsledné aplikace. Implementace aplikace doplňuje již běžící engine přes předdefinované metody. Je proto velmi jednoduché aplikaci upravovat, aniž by bylo nutné zasáhnout do jádra enginu. Na tuto implementaci si engine drží referenci, aby mohl volat její metody. Jeho průběh je popsán v následujících krocích a na obrázku 4.7.

4.3.1 Inicializace

Ve fázi inicializace engine spustí všechny nutné komponenty, tedy LogManagera, knihovnu GLFW s její extenzí GLEW, RenderManagera a nakonec implementovanou inicializaci aplikace.

4.3.2 Smyčka

Dokud se nezavře hlavní okno aplikace, provádí se smyčka, která zahrnuje operace Update a Render.

Update

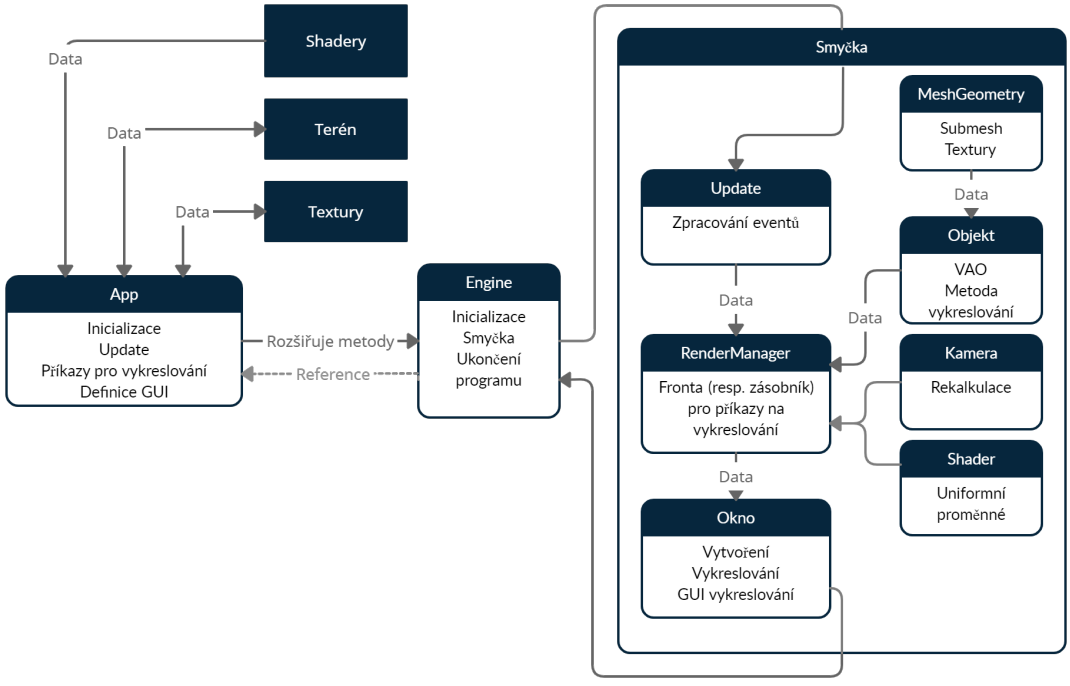
Při updatování se vyprázdní a provede fronta eventů. Také se aktualizují hodiny programu. Nakonec se updatuje implementovaný update aplikace.

Render

Render začíná vykreslováním framebufferu hlavního okna. To je také první příkaz, který se dostane do fronty v RenderManagerovi. Poté následuje přijímání renderovacích příkazů z implementované aplikace. Nakonec se fronta vyprázdní a zpracují se všechny příkazy. Poté probíhá renderování GUI. Nakonec se vymění buffery hlavního okna.

4.3.3 Terminace enginu

Při zavření hlavního okna aplikace engine provede vyčištění paměti a terminace všech držených objektů, tedy RenderManagera, implementovanou terminaci aplikací, zničení okna a terminaci objektů vytvořených pomocí GLFW.



Obrázek 4.7: Diagram návrhu aplikace.

4.4 Vzhled aplikace

Aplikace by měla částečně kopírovat styl uživatelských rozhraní v již vyvinutých aplikacích. Uživatel je při spuštění programu uvítán čistou plochou s hlavním menu nabízejícím jednotlivé kolonky pro operace se soubory, zobrazenými okny a pomocnými nástroji či nastavení editoru a pomocným dialogem.

4.4.1 Dialogová okna

Každé okno, které upravuje objekty nebo nastavení editoru, je v reálném čase přístupné. Tato okna jsou dockovatelná, pohyblivá a škálovatelná.

4.4.2 Modální okna

Okna, ve kterých se vytváří, načítají nebo ukládají objekty, jsou modální. To znamená, že uživatel nemůže zasáhnout do zbytku aplikace při interakci s tímto oknem, aby případně nezměnil data. Rozpoznat takové okno je jednoduché, protože při jeho otevření zšedne nepoužitelné pozadí. Nejsou dockovatelná a některá z nich nejsou pohyblivá. Jsou takto navržena okna pro iniciální tvorbu terénů, managování scény, přidávání shaderů do editoru, ukládání terénů a textur a chybový dialog.

4.4.3 Okno scény

Scéna je zobrazována v paletovém okně scény (obr. 4.8). Toto okno zobrazuje texturu framebufferu scény a upravuje poměr stran. Podle toho se rekalkuluje kamera scény. Poměr stran se také dá nastavit na konkrétní hodnoty (16:9, 4:3, 1:1), škálovatelnost okna je pak omezena jen na jeho šířku. Je dockovatelné. Jeho podokna jsou následující:

Okno pro generování výškových map

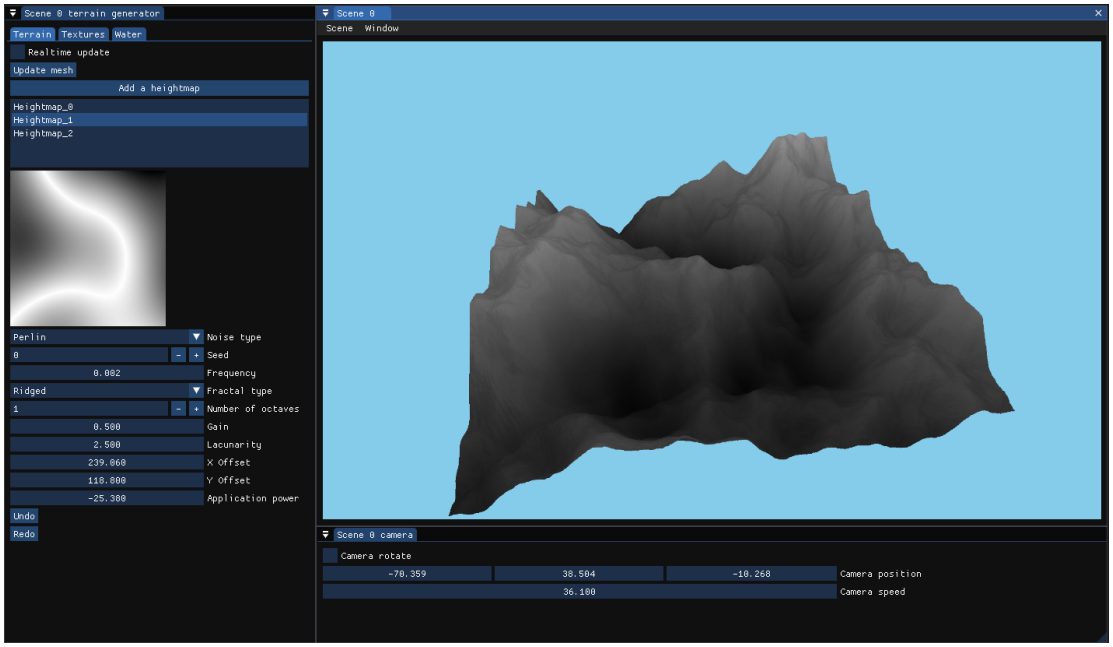
Paletové okno pro generaci výškových map obsahuje listbox s tlačítkem pro přidávání výškových map (obr. 4.8). Mapy se z komboboxu dají mazat a přesouvat do jiných pořadí. Kliknutím pravým tlačítkem na mapu má uživatel možnost ji uložit, načíst nebo upravit. Pro ukládání a načítání se otevře modální okno. Pro úpravu textury se otevře paletové okno s možnostmi přebarvit výškovou mapu nebo postupně zvyšovat hodnoty. Je dostupný čtvercový a kulatý štětec.

Okno pro generování textur

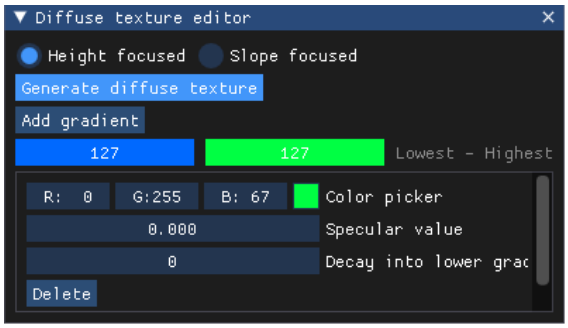
V dalším okně jsou zobrazeny textury výškové mapy, normálové mapy a je zde možné vytvořit difuzní mapu pomocí tlačítka. To otevře okno, ve kterém uživatel volí, zda chce obarvovat terén pomocí výšky nebo sklonu plošek (obr. 4.9). Uživatel přidává, mění rozsah a volí barvy gradientů. Kliknutím pravým tlačítkem na jednotlivé textury může uživatel zvolit, kterou texturu chce zobrazovat na terénu, ukládat v průběhu práce nebo upravit. Upravitelné textury v tomto okně jsou difuzní a spekulární. Pro jejich úpravu se otevře paletové okno se samotnou texturou a nástroji pro výběr barvy.

Okno pro generování vodních ploch

Toto okno zobrazuje výškovou mapu a tlačítka pro rekalkulaci a resetování. Kliknutím na výškovou mapu se vytvoří simulace proudu vody, která steče do lokálního minima a vytvoří zde vodní plochu. Tato skutečnost je zobrazena RGBA texturou, která překrývá výškovou mapu. Na té se zobrazuje výsledná mesh vodní plochy, kde modrá barva je část, která je plně viditelná, zatímco průhledná modř znázorňuje části meshe, které jsou skryté pod terénem nebo ho protínají.



Obrázek 4.8: Okno scény, okno pro generování výškových map (vlevo).



Obrázek 4.9: Okno pro generování difuzních a spekulárních textur.

Kapitola 5

Realizace

Tato kapitola popisuje implementaci objektů a algoritmů aplikace a enginu v C++. Jsou zde zmíněny i alternativní a vylepšující metody, které nejsou v aplikaci implementovány, ale řeší některé její problémy. Jsou zde také demonstrovány výsledky aplikace.

5.1 Řešení metody main

Implementace celé aplikace je řešena v main.cpp a případně dalších souborech, které přímo nesouvisí s enginem. Proto engine vlastní hlavičkový soubor main.h, ve kterém se metoda main nachází.

Aby nedocházelo k duplikaci metody *main*, implementovaná aplikace ji neobsahuje. Pouze pomocí implementované subclassy třídy *App*, kterou vrací metoda *CreateApp()*, přetěžuje virtuální metody nacházející se v *App.h*, které následně volá engine. Díky tomu je ale velice jednoduché implementaci aplikace rozšířit a upravit, aniž by se musel engine upravovat. Je to vhodný model pro vytváření addonů a pomocných skriptů.

```
1  //App.h
2  class App
3  {
4  public:
5          App() {}
6          ~App() {}
7          virtual void Initialize() {}
8          virtual void End() {}
9
10         virtual void Render() {}
11         virtual void Update() {}
12
13         virtual WindowProperties GetWindowProperties()
14         { return WindowProperties(); }
15         virtual void MyGUIRender() {}
16 };
```

5.2 Engine

Instance enginu je rešená jako singleton, který je získáván metodou *GetInstance()*. Zaručuje, že je spuštěná pouze jedna aplikace na tomto enginu. Drží si ukazatel na aplikaci pro referenci, aby z ní mohl volat virtuální metody.

5.2.1 RenderManager

Tento objekt si drží frontu pro *RenderCommand* ve formě chytrých ukazatelů typu unique. Pomocí metody *Submit(std::unique_ptr<RenderCommand> command)* je možné do fronty přidávat příkazy. Také si drží dva zásobníky pro objekty tříd *Framebuffer* a *Camera*.

RenderCommand

Příkazy pro *RenderManager* jsou definovány pomocí dopředně definovaných tříd.

```
1 class Framebuffer;
2 class Shader;
3 class Camera;
4 class Object;
5 class Texture;
```

Pro každou z těchto tříd jsou následně implementovány příkazy pro přidávání a odebírání z momentálního kontextu. Tyto příkazy mají svoji virtuální metodu *Execute()*, která je podle jednotlivých objektů přetížena. V těchto přetížených metodách se předávají do GPU uniformní proměnné a data pro vykreslování. Do konstruktoru pro jednotlivé příkazy se předávají *std::weak_ptr*, aby se nezvyšoval referenční počet *std::shared_ptr*. Je implementováno macro *EDITOR_SUBMIT_RC*, díky kterému je podávání *RenderCommand* mnohem jednodušší a přehlednější.

Pro všechny třídy používané v *RenderCommand* jsou implementovány bindovací funkce, které mění momentální kontext.

5.2.2 Objekty

Objekt je definován třídami *Object* a *MeshGeometry*, na kterou si drží ukazatel. *MeshGeometry* je sestavena z $1 - N$ submeshí, které si zvlášť pamatují informace o počátečních indexech a o identifikačních číslech textur.

5.2.3 Vykreslování

Vykreslování probíhá skrze *RenderManager*. Vyprázdněním fronty pro *RenderCommand* jsou do GPU pomocí *OpenGL* předána veškerá potřebná data. Je implementován shader, který pomocí Phongova osvětlovacího modelu zobrazuje objekty. Kvůli schématu submeshí v geometriích jsou jednotlivé plošky objektů vykreslovány pomocí funkce *glDrawElementsBaseVertex*.

5.3 Aplikace

V souboru *main.cpp* se nachází implementace celého GUI realizovaného pomocí knihovny *ImGui*. Dále jsou implementovány třídy *Terrain* a *Scene*, pomocí které si aplikace drží objekty v kontextu.

5.3.1 Šumové funkce

Pro tvorbu šumových funkcí program využívá knihovnu *FastNoise Lite* [27]. Tato knihovna obsahuje metody určené ke generování již popsaných šumů. Podle rozlišení terénu je možné vygenerovat šumy, které se následně aplikují na vytvořenou geometrii.

Tvorba výškových map

Metoda *CreateHeightMap* přebírá informace o počátečním bodě a seedu šumové funkce a následně vypočítá hodnoty pro výškovou mapu. Každá mapa si pamatuje svých posledních 11 stavů a pomocí undo/redo je možné mezi nimi přepínat.

Aplikace výškových map na terén

Terrain drží pole výškových map, které během rekalkulace výšek postupně sčítá na své souřadnice pomocí metody *ApplyHeightmap*. Protože si terén pamatuje svoje počáteční vrcholy, rekalkulace je efektivní a bezeztrátová, ale náročná na paměť. Rekalkulace může probíhat i v reálném čase, nicméně tento způsob není doporučený pro terény větší než 129x129.

Optimalizace výškových map

Jedinou možností aplikování vytvořených výškových map na geometrii terénu je přičítat její hodnoty pomocí lineárních kombinací. Další možností, která implementována není, je podle vytvořených map násobit geometrii, resp. jednotlivé výškové mapy, a tak vytvořit masky, podle kterých se výšková mapa aplikuje.

5.3.2 Terén

Třída *Terrain* rozšiřuje základní definici třídy *Object*. Pro účel terénu je počet subgeometrií roven 1. Ukládá si veškeré souřadnicové parametry, počet vrcholů a pamatuje si svůj nejnižší a nejvyšší bod. Na všechny typy terénových textur vlastní *Terrain* ukazatele. Jejími podtřídami jsou třída *Gradient*, pomocí které je terén obarvován, a třída *WaterPlane*, která taktéž dědí z třídy *Object* a znázorňuje vodní plochy terénu.

Diamond-square algoritmus

Diamond-square algoritmus je implementován iterativně s použitím dvou front. Jedna fronta slouží pro diamantové a jedna pro čtvercové kroky. Když je index navštívený, uloží se do visited pole. Cyklus začíná diamantovými kroky, které plní frontu čtvercových kroků. Čtvercové kroky naopak plní frontu diamantových kroků, nicméně po vyprázdnění fronty čtvercových kroků se kroková vzdálenost zmenší o polovinu a znásobí se mitigace pro menší rozptyl.

Random fault algoritmus

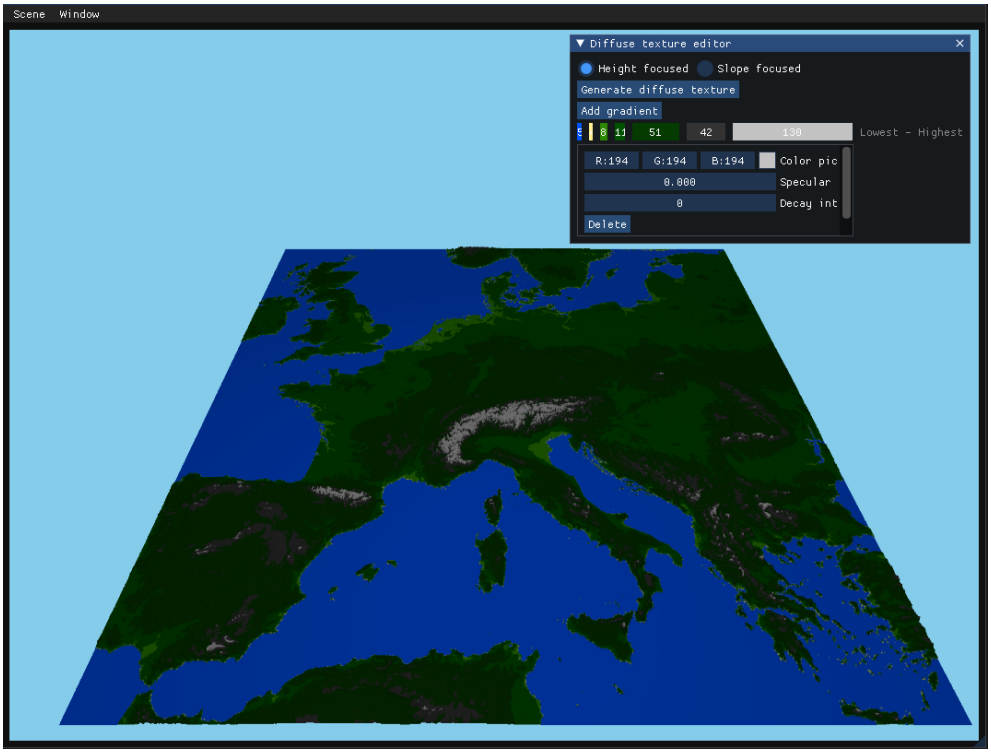
Z uniformního rozdělení algoritmus vybere pro každou iteraci dva body, které proloží přímkou. Funkční hodnoty této přímky se uloží do pole, podle kterého se kontroluje výška jednotlivých bodů. Pro všechny body, které jsou nad (resp. pod) přímkou, se náhodně přičte kladná nebo záporná hodnota z normálního rozdělení znásobená mitigací. Na opačné straně přímky se přičte hodnota záporná té první. Po každé iteraci se mitigace znásobí pro menší rozptyl.

Rekalkulace vrcholových normál

Normály vrcholů jsou přepočítávány pomocí algoritmu, který zkontroluje plochy, ve kterých se vrchol nachází, přepočítá jejich normály a sečte je. Výsledný normalizovaný vektor tvoří normálu vrcholu.

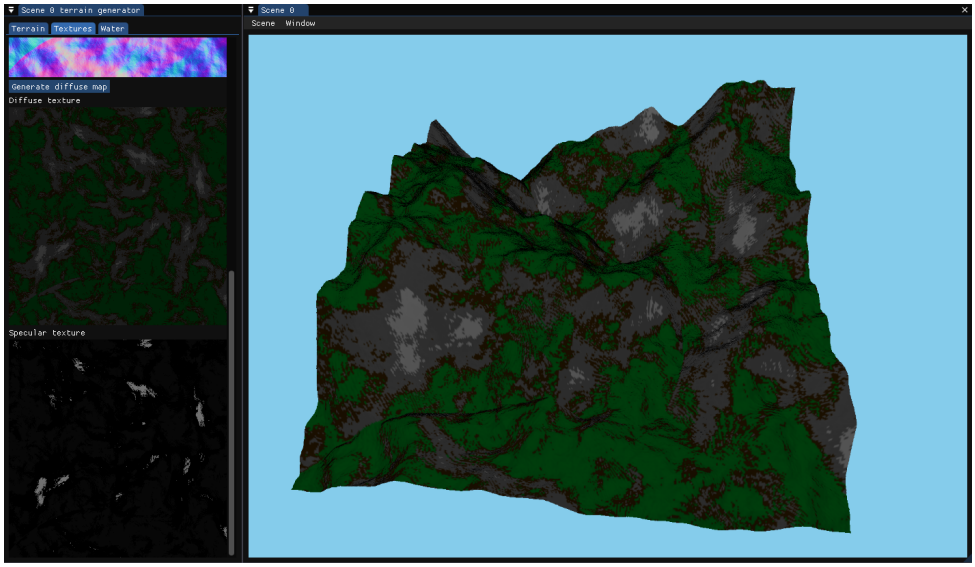
Obarvování difuzní textury

Jsou implementovány dva druhy obarvování. Prvním z nich je obarvování podle výšek. Algoritmus prohledá pole gradientů a postupně obarvuje body difuzní textury podle výškové mapy a intervalů výšek gradientů. Je implementované prolínání do nižších úrovní gradientů, které je určeno absolutní hodnotou normálního rozdělení $N(0, 1)$ násobeného vzdáleností, do které může zasahovat. Tato hodnota je odečítána od spodní meze intervalu. Tento typ obarvování je vhodný pro rozsáhlé globální terény s různými typy materiálů.



Obrázek 5.1: Vygenerovaná difuzní textura podle výškových gradientů

Druhým typem obarvování je obarvování podle sklonu plochy. Pomocí kvaternionů algoritmus vypočítá odklon vrcholové normály od vektoru směřujícího vzhůru. Podle toho, zda odklon leží v intervalu gradientu se obarvuje terén. Tento typ obarvování je vhodný pro menší lokální terény se stejným typem materiálu.



Obrázek 5.2: Vygenerovaná difuzní textura podle gradientů sklonu, vlevo náhled textur

Součástí obarvování je i specifikace spekulárních hodnot v jednotlivých gradientech. Proto souběžně s difuzní texturou vzniká i spekulární textura.

Ukládání a načítání terénu

V programu je implementováno ukládání a načítání terénu ve formátu *.obj*. Výškovou mapu terénu a jeho ostatní textury je možné uložit pomocí knihovny *stb_image* do formátů *.tga*, *.png*, *.jpg* a *.bmp*. Výšková mapa je normalizovaná. Nevýhodou ukládání výškové mapy oproti formátu *.obj* je, že si textura nedrží výškové informace.

Optimalizace zobrazování terénu

Program vykresluje polygony pomocí *GL_TRIANGLES*. Optimalizací této metody je vykreslování přes *GL_TRIANGLE_STRIP*, která značně zredukuje pole indexů.

Stejně jako u World Machine (obr. 1.4) by jednou z možných optimalizací bylo používat přibližné aproximace terénů realizovaných pomocí LOD[1]. Nejen zobrazování, ale i tvorba komplexního terénu v reálném čase může totiž trvat příliš dlouho. Tato metoda je založena na operacích v CPU.

Existuje také metoda ROAM[2], která je pro změnu založena na operacích v GPU. Metoda je založena na datech uložených v binárních stromech. Načítáme data do takové hloubky, abychom optimalizovali chod programu.

5.3.3 Vodní plochy

Vodní plochy jsou tvořeny pomocí speciální textury *WaterMap*, která definuje rozložení vodních ploch. Generují se podle ní plochy, které mohou utvářet moře či jezera (obr. 5.3).

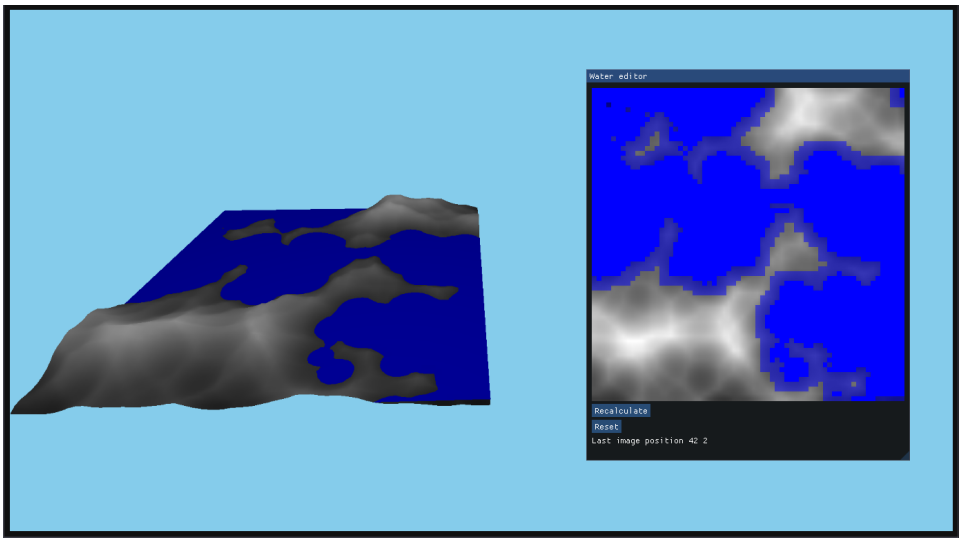
[1] Úroveň detailu (*Level of detail*)
[2] Optimálně adaptující se mesh v reálném čase (*Real-time optimally adapting mesh*)

Obarvování *WaterMap*

Metoda obarvování spočívá v iterativním prohledávání do hloubky přes všechny sousedící body. Pokud metoda narazí na lokální minimum ve výškové mapě terénu, obarví bod modrou barvou. Pokud narazí na vodní hladinu, obarví bod nad vodní hladinou modrou barvou. Současně je kontrolován nejvyšší bod, který je obarvený. Podle něj jsou obarveny všechny body, které jsou nižší nebo stejně vysoko na výškové mapě terénu, ale nemají vodu.

Generování plochy

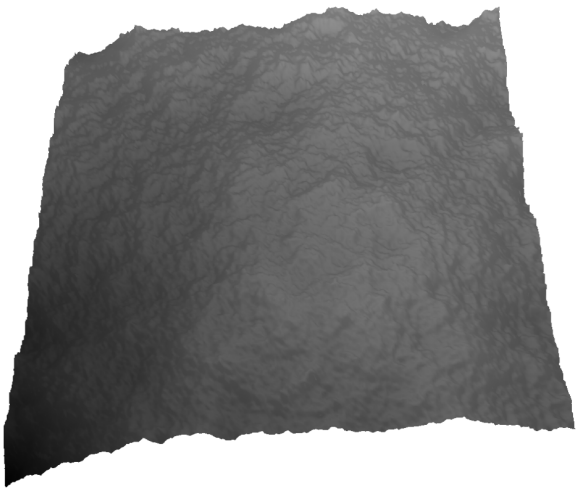
Plocha je následně generovaná algoritmem, který prohledává všechny body textury *WaterMap*. Když narazí na modrou barvu, pomocí prohledávání do šířky prohledá všechny modré sousedy a oindexuje je. Zároveň při prohledávání do šířky kontroluje nejvyšší obarvený bod a výšku celé skupiny nastaví na tuto hodnotu. Poté projde mřížku ještě jednou a podle toho, zda má bod index, sestaví mřížku podobně jako u generování meshe terénu. Výsledné plochy jsou součástí jedné meshe. Indexace je připravena pro schéma submeshí třídy *MeshGeometry*, nicméně rozdělení do submeshí není implementováno.



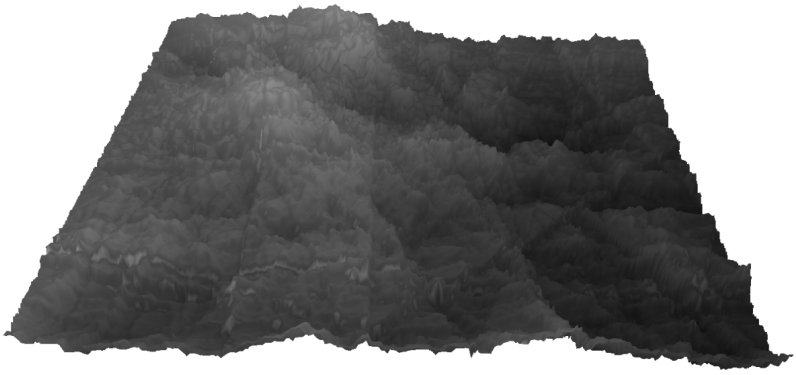
Obrázek 5.3: Vygenerovaná vodní plocha podle textury *WaterMap* (vpravo).

5.4 Výsledky

Složit multifraktální terény pomocí šumových funkcí je poměrně jednoduché a intuitivní oproti používání algoritmů, nad kterými nemáme plnou kontrolu. U diamond-square algoritmu (obr. 5.4) je poměrně složité najít "zlatý střed"hodnot, abychom z algoritmu dostali taková data, jaká chceme. Narozdíl od diamond-square je random fault lépe ovladatelný, ale nepodává tak uvěřitelné výsledky. V následné kombinaci s šumovými funkcemi však na uvěřitelnosti přibývá.



Obrázek 5.4: Terén 257x257 vytvořený pomocí algoritmu diamond-square, posun v měřítku 1/6, Hurstův koeficient 0,6.



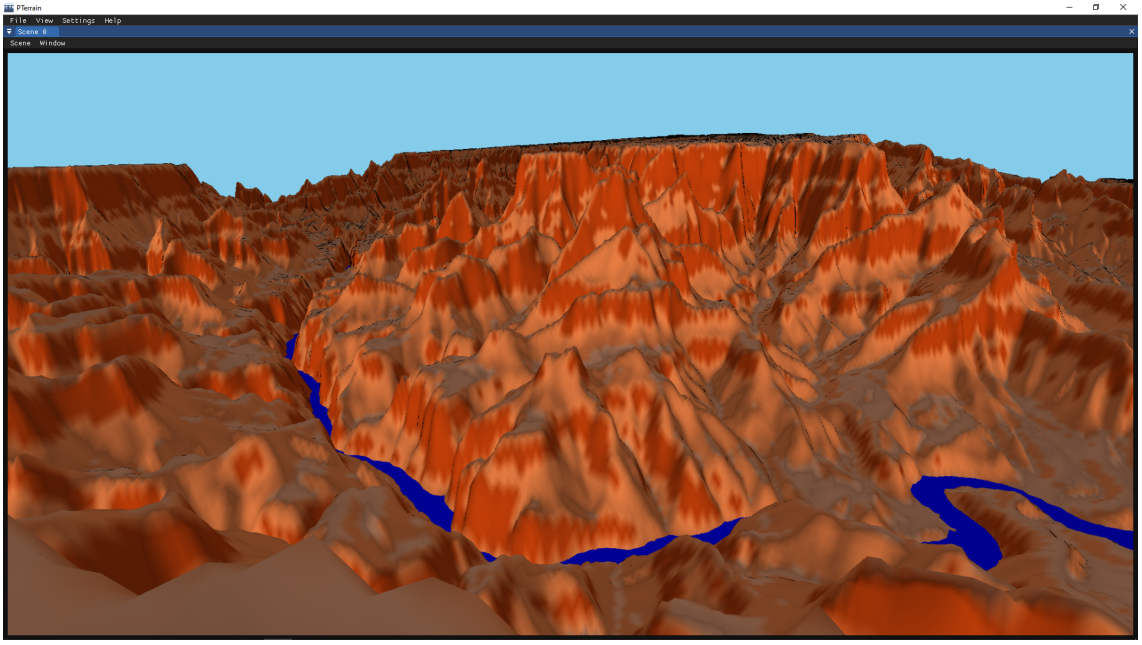
Obrázek 5.5: Terén 257x257 vytvořený pomocí algoritmu random fault, 3600 iterací, Hurstův koeficient 0,43.

5.4.1 Realismus

Pro porovnání s reálnými krajinami byly zvoleny tři typy terénů. První z nich je s načtenou výškovou mapou národního parku Grand Canyon v Arizoně (obr. 5.6). Pohled na scénu je z východního cípu řeky Colorado. Srovnání je s reálnou fotografií z nedalekého vyhlídkového místa Desert View Point (obr. 5.7).

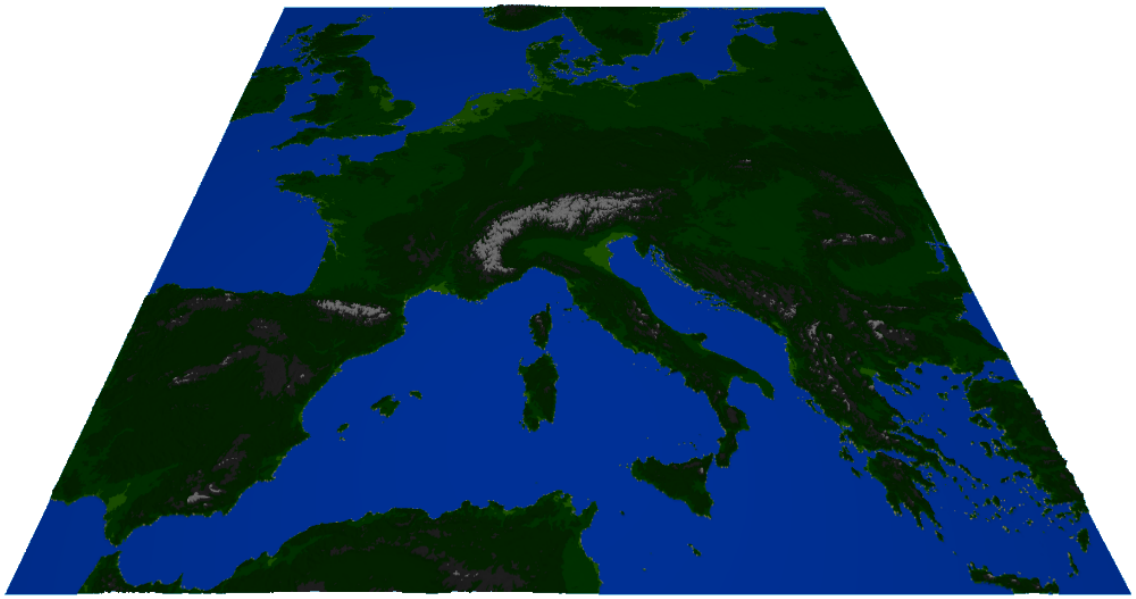
Pro druhé porovnání je využito terénu Evropy ve velkém měřítku z obrázku 4.2 (obr. 5.8). Srovnání je se satelitním snímkem Evropy, který je ortografický (obr. 5.9).

Pro třetí porovnání byl zvolen terén generovaný editorem. Jedná se o terén 513x513 vygenerovaný pomocí multifraktálů. Verze na obrázku 5.10 je obarvená podle sklonu, zatímco verze na obrázku 5.11 je obarvená podle výšek. Pro srovnání přišla vhodná fotografie 5.12 Popradského plesa ve Vysokých Tatrách na Slovensku.



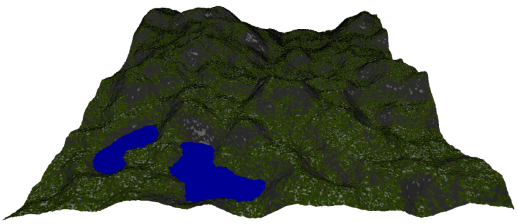
Obrázek 5.6: Vygenerovaný Grand Canyon s rozměry 800x800.

Obrázek 5.7: Pohled z Desert View Point [28].

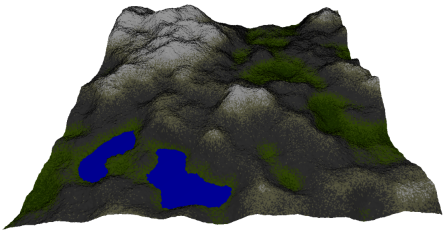


Obrázek 5.8: Vygenerovaný terén Evropy s rozměry 1025x1025.

Obrázek 5.9: Satelitní snímek Evropy.



Obrázek 5.10: Terén 513x513 obarvený podle sklonu.



Obrázek 5.11: Terén 513x513 obarvený podle výšek.

Obrázek 5.12: Pohled na Popradské pleso v Tatrách na Slovensku [29].

Závěr

Tato bakalářská práce seznámila čtenáře s principy procedurálního generování, které budou v moderním světě hrát čím dál důležitější roli, protože se jeho možné aplikace stále rozšiřují. Bylo prozkoumáno, že kromě tvorby čísel se tato disciplína zabývá metodami spojenými s texturováním a samotným modelováním. Pomocí fraktálů se teorie rozšířila o šumové funkce, které jsou esenciálním faktorem procedurálního generování terénů. Kromě šumových funkcí byly v práci vysvětleny i principy *fBm* a s ním související algoritmy: metoda přesouvání náhodného bodu, algoritmus *diamond-square* a na závěr metoda náhodných poruch.

Na těchto teoretických základech byl pro tvorbu virtuálních světů a jejich úpravy vyvinut editor v jazyce C++ za pomoci grafické knihovny *OpenGL*, knihovny pro tvorbu šumových funkcí *FastNoise Lite* a knihovny grafického prostředí *ImGui*. Byl navržen engine, aplikace samotná, grafické rozhraní a a zároveň byly navrženy algoritmy spojené s tvorbou terénů a jeho textur. Uživatel v editoru může tvořit vlastní terény pomocí šumových funkcí nebo algoritmů *diamond-square* a *random fault*. Byl vytvořen obarvovací systém pro difuzní textury a spekulární textury. Nad všemi tvořivými prvky terénu má uživatel absolutní kontrolu. Také je implementován systém pro tvoření vodních ploch závisejících na již existujícím terénu, který simuluje usazování vody v lokálních minimech. Terény vytvořené v editoru je možné následně ukládat ve formátu *.obj* a importovat zpět do editoru nebo do jiných programů.

Program má určité nedostatky. Nejsou podporovány obdélníkové terény kvůli špatné kompatibilitě textur. Navíc je aplikace limitovaná pamětí, veškerá práce s většími terény (např. nad rozměry 2049x2049) je časově náročná. Algoritmus pro tvorbu vodních ploch je pro větší terény také příliš pomalý z důvodu zdlouhavého procházení mřížky terénu. Možným vylepšením by bylo dynamičtější programování a ukládání průběžných hodnot při tvorbě terénů. Pokud by byla známa místa, která jsou naplněna vodou nebo na kterých se nacházejí lokální minima, bylo by možné použít efektivnější algoritmy pro hledání cest, např. A*. Vodní plochy nemohou být texturované jinak, než určuje aplikace. Navíc nemohou být importovány. Formát pro ukládání *.obj* je velice objemný. Sice drží informace o všech vrcholech, nicméně pro mřížkový terén, řešený v této aplikaci, jsou všechny údaje kromě rozměrů, vzdáleností vrcholů a výšek vrcholů zbytečné.

Silnou stránkou aplikace je vykreslování a přesné upravování terénů v reálném čase. Díky tomu je vytváření geometrií velice rychlé a přehledné. Dalším pozitivem je jednoduché a intuitivní ovládání grafického rozhraní. Uživateli je umožněno rychle ukládat textury, se kterými se setká během práce. Aplikace by díky těmto vlastnostem mohla být vhodná pro indie vývojáře videoher nebo pro rychlou tvorbu scén, např. do stavebního inženýrství.

Bibliografie

1. EBERT, David S; MUSGRAVE, F Kenton; PEACHEY, Darwyn; PERLIN, Ken; HART, John C; WORLEY, Steven. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003.
2. FREIKNECHT, Jonas; EFFELSBERG, Wolfgang. A survey on the procedural generation of virtual worlds. *Multimodal Technologies and Interaction*. 2017, roč. 1, č. 4, s. 27.
3. PARBERRY, Ian. Designer worlds: Procedural generation of infinite terrain from real-world elevation data. *Journal of Computer Graphics Techniques*. 2014, roč. 3, č. 1.
4. ČMOLÍK. *Procedural modeling of 2D and 3D objects*. 2020.
5. SHORT, Tanya; ADAMS, Tarn. *Procedural generation in game design*. CRC Press, 2017.
6. MANDELBROT, Benoit B; MANDELBROT, Benoit B. *The fractal geometry of nature*. WH freeman New York, 1982.
7. WIKIPEDIA CONTRIBUTORS. *Fractal — Wikipedia, The Free Encyclopedia* [`https://en.wikipedia.org/w/index.php?title=Fractal&oldid=1000058057`]. 2021. [Online; accessed 14-January-2021].
8. (PROTIOUS), George. *Mandelbrot black itr20*. 2008. Dostupné také z: `https://en.wikipedia.org/wiki/File:Mandelbrot_black_itr20.png#filelinks`.
9. FIBONACCI. *Kuch curve*. 2007. Dostupné také z: `https://commons.wikimedia.org/wiki/File:Koch_curve.svg#/media/File:Koch_curve.svg`.
10. JPB06. *Terragen, Geocontrol, Vue, World Machine + Blender?* [`https://blenderartists.org/t/terragen-geocontrol-vue-world-machine-blender/507207`]. 2011. [Online; accessed 14-January-2021].
11. SOFTWARE, PLANETSIDE. *Rendering in Terragen 4 v3*. 2019. Dostupné také z: `https://planetside.co.uk/terragen-feature-tour/`.
12. SOFTWARE, World Machine. *Terrain square 2*. 2021. Dostupné také z: `https://www.world-machine.com/features.php`.
13. LAGAE, A.; LEFEBVRE, S.; COOK, R.; DEROSE, T.; DRETTAKIS, G.; EBERT, D.S.; LEWIS, J.P.; PERLIN, K.; ZWICKER, M. A Survey of Procedural Noise Functions. *Computer Graphics Forum*. 2010, roč. 29, č. 8, s. 2579–2600. Dostupné z DOI: `https://doi.org/10.1111/j.1467-8659.2010.01827.x`.
14. STOLFI, Jorge. *White noise*. 2013. Dostupné také z: `https://commons.wikimedia.org/w/index.php?curid=24614072`.

15. CSÉBFALVI, Balázs. Fast Catmull-Rom Spline Interpolation for High-Quality Texture Sampling. *Computer Graphics Forum*. 2018, roč. 37, č. 2, s. 455–462. Dostupné z DOI: https://doi.org/10.1111/cgf.13375.

16. ARCHER, T. 1 Uses for Noise. In: 2011.

17. WIKIPEDIA CONTRIBUTORS. *Gradient noise — Wikipedia, The Free Encyclopedia* [https://en.wikipedia.org/w/index.php?title=Gradient_noise&oldid=910584247]. 2019. [Online; accessed 15-January-2021].

18. WIKIPEDIA CONTRIBUTORS. *Simplex noise — Wikipedia, The Free Encyclopedia* [https://en.wikipedia.org/w/index.php?title=Simplex_noise&oldid=969175213]. 2020. [Online; accessed 16-January-2021].

19. KŘÍŽ, Jakub. *Multi-Fractal Terrain Generation [online]*. 2019 [cit. 2021-12-27]. Dostupné také z: https://is.muni.cz/th/q6114/. Master's thesis. Masaryk University, Faculty of Informatics, Brno. Vedoucí práce Fotios LIAROKAPIS.

20. WORLEY, Steven. A cellular texture basis function. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, s. 291–294.

21. WIKIPEDIA CONTRIBUTORS. *Fractional Brownian motion — Wikipedia, The Free Encyclopedia* [https://en.wikipedia.org/w/index.php?title=Fractional_Brownian_motion&oldid=997408990]. 2020. [Online; accessed 16-January-2021].

22. MUSGRAVE, F. Kenton. Procedural Fractal Terrains. 2015.

23. ŽÁRA; BENEŠ; SOCHOR; FELKEL. *Moderní počítačová grafika*. Computer Press, 2005.

24. EWIN, Christopher. *Diamond square steps*. Dostupné také z: https://commons.wikimedia.org/w/index.php?curid=42510593.

25. GALIN, Eric; GUÉRIN, Eric; PEYTAVIE, Adrien; CORDONNIER, Guillaume; CANI, Marie-Paule; BENES, Bedrich; GAIN, James. A review of digital terrain modeling. In: *Computer Graphics Forum*. 2019, sv. 38, s. 553–577. Č. 2.

26. "TANGRAM". *"Tangram"*. "2022". Dostupné také z: https://tangrams.github.io/heightmapper/.

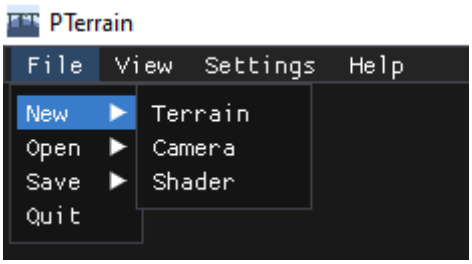
27. *FastNoise Lite* [https://github.com/Auburn/FastNoise]. [N.d.]. Accessed: 2021-01-17.

28. CALDON", "Kristen M. *View north from Desert View Point at the Colorado River*. Dostupné také z: https://www.nps.gov/grca/planyourvisit/desert-view.htm.

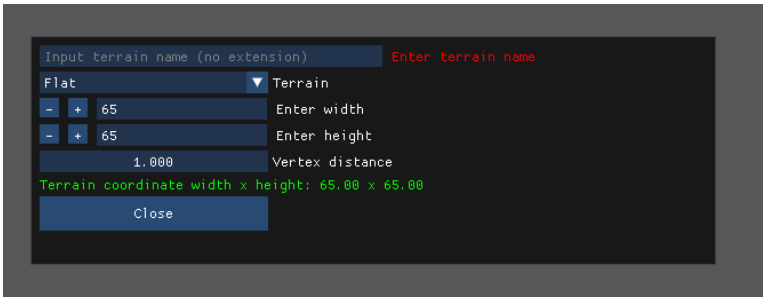
29. "GEJZA". *"Pohled na Popradské pleso a horu Satan"*. "2017". Dostupné také z: https://www.natreku.cz/vht-vystup-na-ostrvu-vysoke-tatry/.

Přílohy

A Ovládání

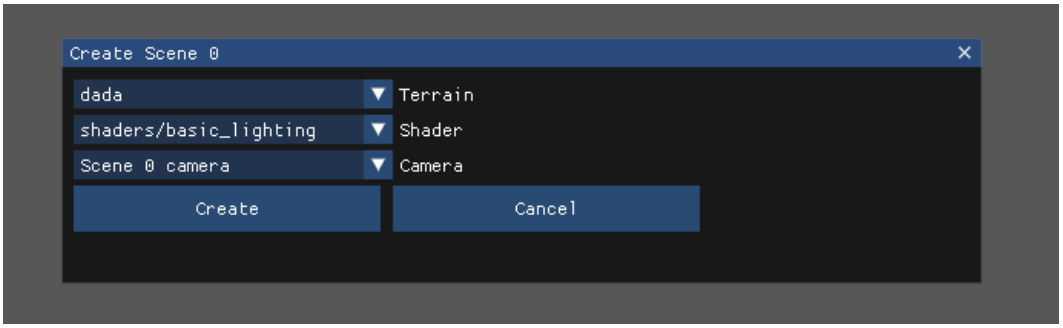


Kliknutím na File se otevře menu, ve kterém lze tvořit, otvírat či ukládat terény a objekty editoru.



Po kliknutí na Terrain se otevře modální okno pro definování parametrů vytvářeného terénu. Po zadání jména se objeví tlačítko Create, v případě, že se už stejné jméno v editoru nevyskytuje. V opačném případě je nutné nový soubor přejmenovat.

Po kliknutí na Create se vytvoří další modální okno s tvorbou scény. Zvolte parametry zobrazovaného objektu, shaderu a kamery. Tyto volby jsou retrospektivně přístupné v menu scény pod názvem Manage Scene. V menu scény Window zobrazte okna pro generování terénu a ovládání kamery a dockujte je tažením po obrazovce a přemístěním na ikonu.

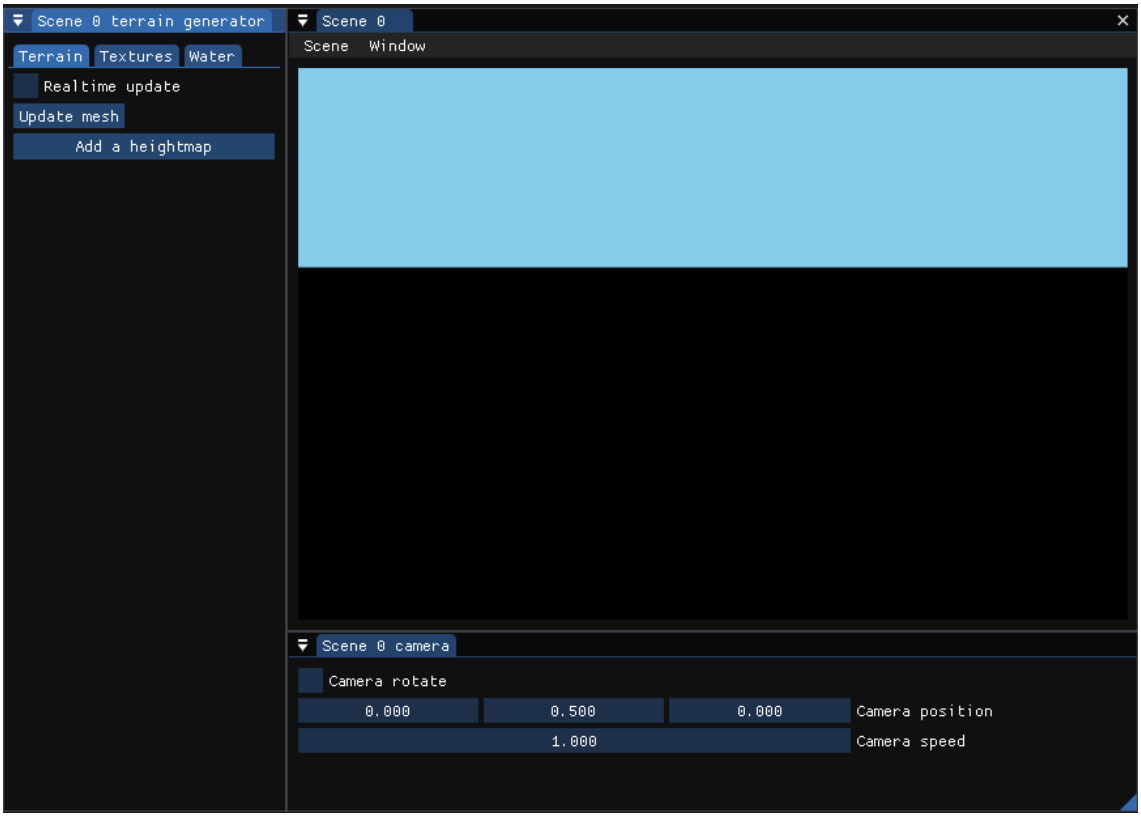


Aplikace je ovládána následujícími kombinacemi:

1. Textura scény - Pravé tlačítko - otáčení kamery.
2. Textura scény - Pravé tlačítko + WSAD - pohyb kamery.
3. Textura scény - Pravé tlačítko + Pg Up - přiblížení kamery.
4. Textura scény - Pravé tlačítko + Pg Down - oddálení kamery.
5. Parametry - Shift + levé tlačítko pro rychlý krok.
6. Parametry - Alt + levé tlačítko pro pomalý krok.

Kliknutím na Add Heightmap v okně pro generaci terénů se přidá šum do listboxu. Je možné na něj kliknout pravým tlačítkem a odebrat ho z listboxu nebo ho přesunout v pořadí. Je nutné pravým tlačítkem kliknout na momentálně zvolený šum.

Kliknutím na Update mesh se aktualizují vrcholy terénu. Při kliknutí na checkbox Realtime update toto tlačítko zmizí a aktualizace se děje automaticky.



B Instalace

Aplikace byla vyvinuta pro Windows ve Visual Studiu 2019. Ke kompilaci a instalaci aplikace jsou nutné následující knihovny:

1. OpenGL 4.6,
2. GLFW v3.3,
3. GLEW,
4. ImGUI docking branch 1.85WP,
5. GLM,
6. FastNoise Lite,
7. stb_image,
8. spdlog.